Examining the Use of Java Technologies for Real-Time in a Prototype U.S. Surface Navy Combat System Application

Naval Surface Warfare Center Dahlgren Division
Fred Weindelmayer and Tim Childress

Disclaimer: The testbed and scenarios described represent notional US Navy Combat System capabilities for technology evaluation purposes.
The OA Surface Application Domain

- Large distributed real-time weapon system applications
- Both soft and hard real-time applications
- Timescale for deadlines typically in the \( O(10) - O(100) \) millisecond range
- Long system life spans, often for several decades
- Consequences of not meeting requirements for time-deterministic behavior could be severe
Standards Support for Real-Time

- **Non-real-time**
  - e.g. business

- **Soft real-time**
  - e.g. C2, display & decision support

- **Hard real-time**
  - e.g. sensor & weapon control

- **Extreme real-time**
  - e.g. signal processing

- **Web Services Technologies**
- **JMS**
- **RTSJ (hard RT)?**
- **Java w/RT GC & AOT ?**
- **CORBA**
- **RT Data Distribution Service**
- **MPI**

**OACE Technologies and Standards**
### Java Considerations in the Real-Time Domain

<table>
<thead>
<tr>
<th>Potential Benefits</th>
<th>Concerns</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Enabling of SOA &amp; Web Services for Cross-Domain &amp; Fn Integration</strong></td>
<td></td>
</tr>
<tr>
<td>- Java provides support for SOA and Web Services technologies such as SOAP, XML, UDDI, J2EE, etc.</td>
<td>Can Java meet real-time, mission-critical, and safety requirements of combat systems?</td>
</tr>
<tr>
<td></td>
<td>- The ability of Java technologies to meet required combat system performance has not yet been proven</td>
</tr>
<tr>
<td></td>
<td>- Insufficient information whether Java will pass Weapons Systems and Explosives Safety Review Board (WSESRB) certification using mainstream features – VLS briefed WSESRB regarding their use of RTSJ with a RAVENSCAR-like profile</td>
</tr>
<tr>
<td><strong>Increased Productivity</strong></td>
<td></td>
</tr>
<tr>
<td>- Large programmer base</td>
<td></td>
</tr>
<tr>
<td>- Standardized APIs</td>
<td></td>
</tr>
<tr>
<td>- Many libraries available to implement commonly required capabilities e.g. string manipulation, data structures, graphics, etc.</td>
<td></td>
</tr>
<tr>
<td>- Automatic memory management (less debugging)</td>
<td>Will Java technologies for real-time deliver the same productivity gains of standard Java?</td>
</tr>
<tr>
<td></td>
<td>- Appropriate use of RTSJ features requires additional training and introduces complexity</td>
</tr>
<tr>
<td></td>
<td>- RT garbage collectors require additional understanding of underlying application memory allocation behavior that is not made apparent by the language – tools not available to support a priori analysis</td>
</tr>
<tr>
<td></td>
<td>- Real-time programming knowledge is required regardless of the language. Java does not alleviate this requirement</td>
</tr>
<tr>
<td></td>
<td>- Some standard Java APIs are incompatible with RTSJ memory management for RT apps; Porting of some APIs has been done, but inconsistent across RTSJ implementations</td>
</tr>
<tr>
<td><strong>Portability of Java Applications</strong></td>
<td></td>
</tr>
<tr>
<td>- &quot;Write once, run anywhere&quot;</td>
<td>Using Java in the real-time domain requires a tighter coupling to the computer platform</td>
</tr>
<tr>
<td></td>
<td>- RTSJ slogan “Write once carefully, run anywhere conditionally”? – not yet proven</td>
</tr>
<tr>
<td></td>
<td>- JVMs implementing RT technologies (e.g., RTSJ, RTGC, AOT, etc) are part of a product ecosystem (JVM, OS, processor), currently having limited availability</td>
</tr>
<tr>
<td></td>
<td>- May require additional standards-related guidance for OS, e.g. which scheduling algorithms must be supported, minimum clock resolution</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product availability</th>
<th>Will Java products for real-time have the same availability?</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Java is widely supported with many products available for virtually any computing platform</td>
<td></td>
</tr>
<tr>
<td>- Many support products available that support Java, including development environments, middleware, etc.</td>
<td>- Vendors are still limited in the diversity of platforms that they support; very few supported products – several available in beta.</td>
</tr>
<tr>
<td></td>
<td>- Differences between Java products for RT are significant at this time – cost of switching products may be significant if a computing platform port is required.</td>
</tr>
<tr>
<td></td>
<td>- Significantly smaller market for RT to drive development of Java and support products</td>
</tr>
</tbody>
</table>
Java Technologies for Real-Time

- **Standard Java**
  - Sun, IBM, and BEA J2SE 5.0 JVMs
  - Can be used for some real-time applications if features that impact determinism are restricted, e.g.
    - Employ memory pools to reduce / eliminate dynamic allocation of objects
    - Constrain use of standard Java libraries
    - Avoid multi-threading to eliminate possible priority inversion

- **The Real-Time Specification for Java (RTSJ)**
  - Sun Java RTS and IBM J9 RT JVM (J2SE)
  - Apogee (J2ME)

- **Real-Time Garbage Collectors**
  - Implementations exist for both Standard Java and RTSJ products
    - Metronome GC (IBM), Lund GC (Sun), Deterministic GC (BEA)

- **“Java-Like” Virtual Machines with Real-Time GC**
  - Aonix PERC and AICAS Jamaica VMs
Objectives of the Experiment

- Obtain information on which technologies, and which subsets of features within them, are appropriate for different regions of the real-time Navy Enterprise application space
  - For example: Is there a class of soft real-time applications for which real-time garbage collection is sufficient, without having to use RTSJ-specific features (even if using an RTSJ JVM)?

- Gain insight on appropriate techniques and design patterns for application of real-time Java technology
  - Languages typically provide many alternative ways to implement solutions, but which are best suited to the problems we encounter?

- Identify possible deficiencies in existing real-time Java technologies for Navy real-time applications
  - Such knowledge could be applied toward the development of a Mission-Critical Java standard

- Gain sufficient insight on RT Java usage to determine how standards might be changed in support of Navy RT requirements
Metrics of Interest

- **Timeliness**
  - Degree to which application meets real-time deadlines
  - Track generator will simulate a variety of different message traffic patterns, allowing us to evaluate timeliness under different load scenarios

- **Jitter**
  - Difference between longest and shortest observed latencies
  - Lower jitter means more consistent real-time behavior and consequently more predictability

- **Performance**
  - Performance impacts which deadlines can be feasibly met with the technology

- **Relevant characteristics associated with application development:**
  - Portability
  - Maintainability
  - Scalability
  - Ease of development
Evaluation Description
Primary Goal: Investigate how well Java applications integrate into and perform in a distributed combat system environment with legacy components.
C++ and standard Java implementations of OACE Auto Special prototype using same high-level design

“Reasonable decisions” principle for Java implementation – e.g. memory pools for fixed sized objects

Identical test harness, procedures, and scenario

Instrumentation points inserted at identical points in the processing

Language Implementations:
- C++: GCC 3.4.4
- Standard Java: J2SE 5.0 (various vendor JVMs)

Computing Platform:
- Intel Xeon 3.06 GHz, 4 CPUs, 1GB RAM
- Red Hat Linux 3.4.3-9.EL4 with RT patches
- NDDS 4.0

*Lesson learned from Aegis Open Architecture*
Sensor-To-Shooter Instrumentation

**T1** - Valid Time of Sensor Kinematic Data
**T2** - Time Track Data Received at Service
**T3** - Time Track Data Sent from Service
**T4** - Time Track Data Received at Consumer
**T5** – Time Track Data Processed at Consumer

**T6** - Valid Sensor A/S Message Initiated
**T7** - A/S Message Received at Consumer
**T8** - A/S Engagement Request Sent
**T9** - A/S Engagement Request Received
**T10** - A/S Engage Order Sent

**T11** - A/S Periodic Processing Start
**T12** - A/S Periodic Processing Complete

*Tn* – Instrumentation point in OACE Prototype
*Tm* – Instrumentation point used for Java/C++ Comparison

Simulation  AAW C²  Fault tolerant and/or Scalable  C++ & Java

6/20/2006  Slide 12
Auto Special Activity Diagram

Discovery Event
Peer Monitor
Real-Time Track Processing
DDS Callback
Sensor Declared A/S Track Proc
Doctrine Statement Processing
Ownship Data Processing
Engagement Status Processing
Periodic Doctrine Review

Validate Event
Route Event
Update Peers
Update Local Track Datastore
Process Track
Process Track
Perform A/S Eval on Sensor Declared Track
Send A/S Engagement Request
Update Local Doctrine Datastore
Update Local Ownership Datastore
Update Local Engagement Datastore
Start Review
Perform A/S Eval on all Tracks in Datastore
End Review
Send SM2 Engagement Request

Tn – Instrumentation point in OACE Prototype
Tm – Instrumentation point used for Java/C++ Comparison
Scenario

- One hour data collection period for each configuration
- Common test harness & track scenario for C++ and Java

- Six surface tracks updated at 2Hz rate
- Seven pop-up tracks updated at 2Hz rate – included 3 A/S engageable tracks
- 1000 background track load – updated at ½ Hz rate
Results
Data will be provided
Conclusions
Language selection appears to have a greater effect on overall processing time than the language implementation

- C++ performance is highly predictable throughout the test
- Both Java products in this experiment exhibit some degree of unpredictable performance – unknown how they will scale to larger processing loads or how outliers will affect the ability to meet end-to-end performance requirements
- RT garbage collection technology is promising - exhibits fewer outliers that tend to be smaller in magnitude

Cause of Java outliers is currently unknown – candidates include:

- RT Garbage Collection technology is immature
- Garbage collection effects
  - May be able to further optimize application using standard Java features
  - May be able to tune GC products to significantly reduce number or size of outliers
- Just In Time (JIT) compilation
  - May introduce processing delays when a code path is traversed for the first time
- Secondary effects of Java implementation or other language features e.g. thread control
  - May be able to use RTSJ thread control features in combination to reduce effects
- Suitability of high-level design for optimal Java performance
  - Redesign may improve overall performance and enhance understanding of RT design patterns for Java

Predicting and tuning performance will be a key concern when using RTGC and AOT technologies – analysis will require methodologies and tools not currently available
Observations

- Java products for real-time are a potential solution for allowing Java applications to meet Navy deterministic mission needs. A number of products now exist; all are immature at this point.

- Real-Time Garbage Collection technology is generally immature. Because no standards exist for garbage collection, behavior and performance can vary significantly.

- Java technologies for real-time are often tied to the underlying computing infrastructure. Use of Java technologies for real-time may ease code portability, but may also require additional standards-related guidance to support reuse and technology refresh while meeting performance requirements.

- It is not clear at this point how widely-accepted Java technologies for real-time will become. Need to continue working with Java vendors to improve their products’ viability within the Navy (performance, tools, standardization approach, business model, product licensing etc.).

- Additional experiments needed to fully understand performance and its effect on meeting real-time end-to-end system requirements.
Questions?
Backup